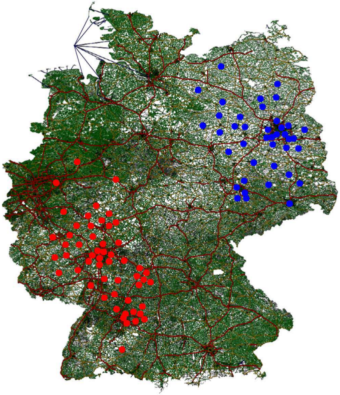
Engineering Time-Dependent Many-to-Many Shortest Paths Computation

Robert Geisberger and Peter Sanders
{geisberger, sanders}@kit.edu, http://algo2.iti.kit.edu/routeplanning.php

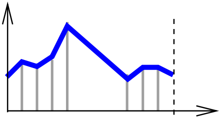
Institute for Theoretical Informatics, Algorithmics II

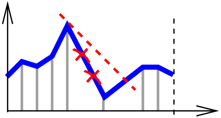
Motivation

- To solve logistics problems, it is necessary to know the travel times between all locations.
- In the time-independent scenario, it is possible to compute a $10\,000 \times 10\,000$ table in 10 seconds.
- But there is no efficient algorithm when travel times depend on the departure time.



Time-Dependent Route Planning





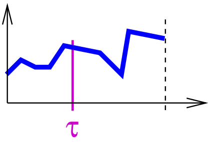
- Edge weights are travel time functions (TTFs)
  - {point in time $\mapsto$ travel time period}
  - piecewise linear
  - FIFO-property – waiting does not help
- Kinds of queries to a table cell
  - Time query:
    Earliest arrival depending on a given departure time
  - Profile query:
    travel time profile:
    {departure time $\mapsto$ travel time period}

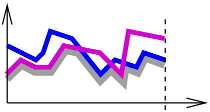
Operations on Travel Time Functions

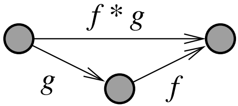
We need three operations

- Evaluation: $f(\tau)$ — "O(1)" Time
- Merging: $\min(f, g)$ — $O(|f| + |g|)$ Time
- Chaining: $f * g$ (f "after" g) — $O(|f| + |g|)$ Time

Note: $\min(f, g)$ and $f * g$ have $O(|f| + |g|)$ points each.
$\Rightarrow$ Increase of complexity







Profile Dijkstra algorithm

- implemented using merging and chaining on TTFs
- label-correcting

Related Work

Time-dependent Profile Dijkstra

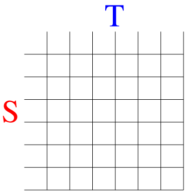
- SSSP computation not feasible on large graphs.

Time-dependent Time Dijkstra for set of departure times

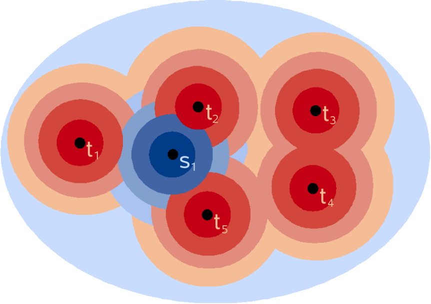
- Provides no approximation guarantee.
- Requires to much space as times are stored redundantly.

Point-to-point speed-up techniques:

- Faster than Dijkstra.
- Still require quadratic time and space to compute table.



Related Work



Static Many-to-Many Algorithm

- use bidirected and non goal-directed speedup technique
- for each $t \in T$, perform backward search, store search space entries $(t, u, \overleftarrow{\delta}_t(u))$
- arrange search spaces: create a bucket for each u
- for each $s \in S$, perform forward search, at each node u, scan all entries $(t, u, \overleftarrow{\delta}_t(u))$ and compute $\overrightarrow{\delta}_s(u) + \overleftarrow{\delta}_t(u)$, minimum over all candidate nodes u is shortest paths distance.

Problems in Time-Dependent Scenario

Karlsruhe Institute of Technology

- Cheap operations in the static scenario (add, min on integers) are mapped to expensive operations on travel time functions (TTF).

  $\Rightarrow$ use a more sophisticated approach to skip a lot of these operations

- The computation of table of TTFs for all $S \times T$ require inherently $\Theta(|S| \cdot |T|)$ time and space.

  $\Rightarrow$ Redefine the problem to the implementation of a query interface:

  - time query: $(s, t, \tau) \mapsto$ earliest arrival time
  - profile query: $(s, t) \mapsto$ travel time profile

Our Contributions

Straightforward:

- Usage of a bidirected and non-goaldirected time-dependent speedup technique: Time-dependent Contraction Hierarchies (TCH) [ALENEX'09, SEA'10].
- Compute forward and backward profile search spaces.
- Intersection of search spaces results in sought after travel times.

More complicated:

- Compute additional information to speedup intersection of search spaces.
- Reduction of search spaces to relevant nodes.
- Usage of approximate TTFs together with approximation guarantees.

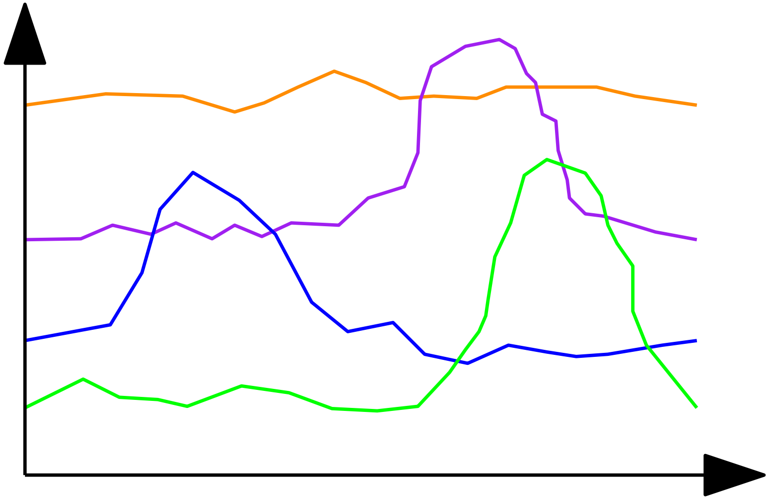
Intersect Algorithm

- Precompute and store forward/backward search spaces F_s / B_t
  $\Rightarrow \Theta(|S| + |T|)$
- Answer by intersecting F_s and B_t.
- Use min/max values and lower/upper bounds for pruning.

F_s
$(7, f_7)$
$(10, f_{10})$
$(11, f_{11})$
$(16, f_{16})$

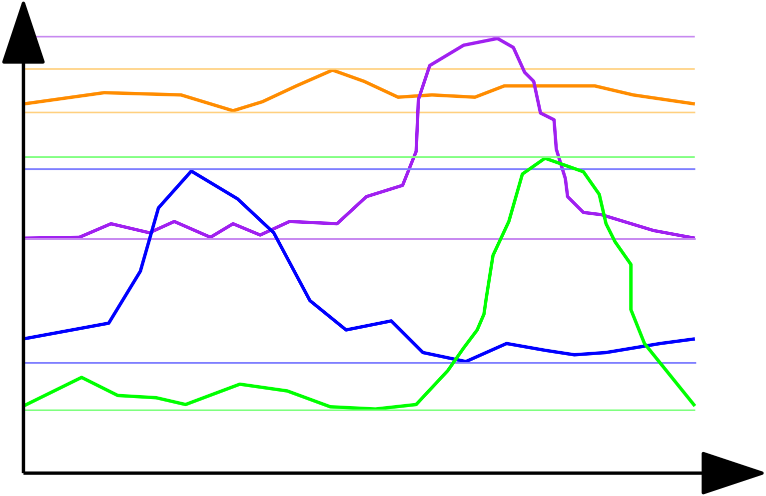
B_t
$(10, g_{10})$
$(11, g_{11})$
$(12, g_{12})$
$(15, g_{15})$

Pruning



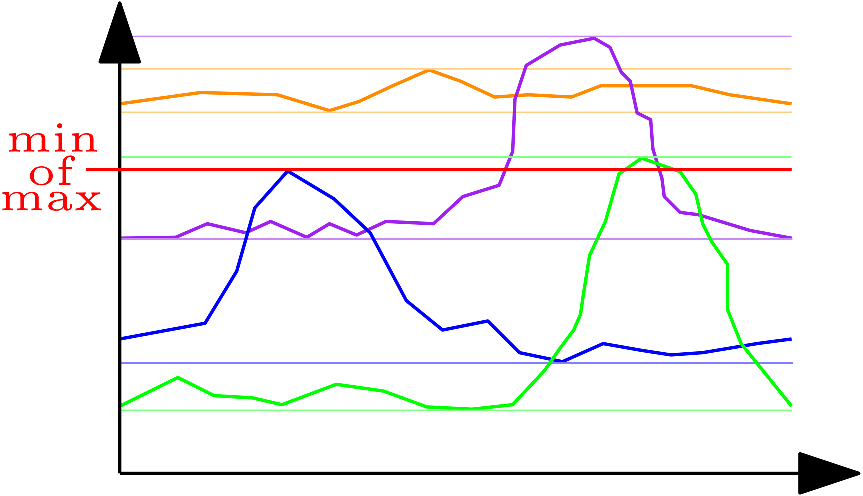
Pruning

Very cheap operations on minimum/maximum of TTFs



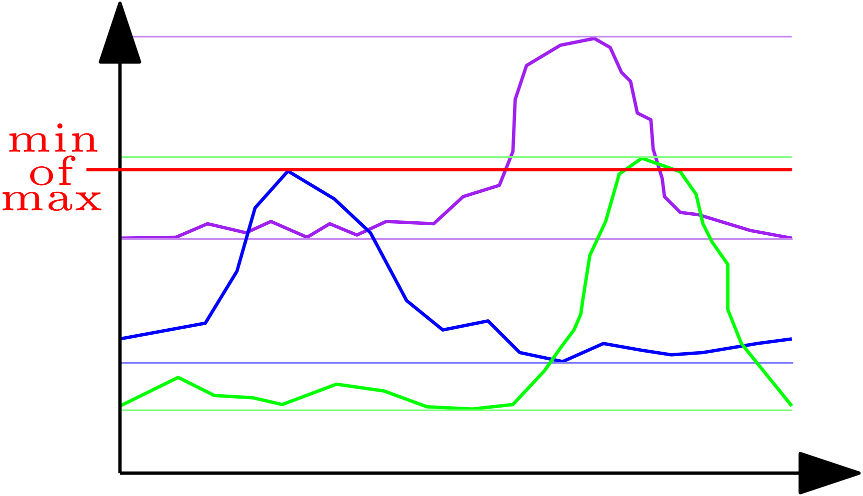
Pruning

Very cheap operations on minimum/maximum of TTFs



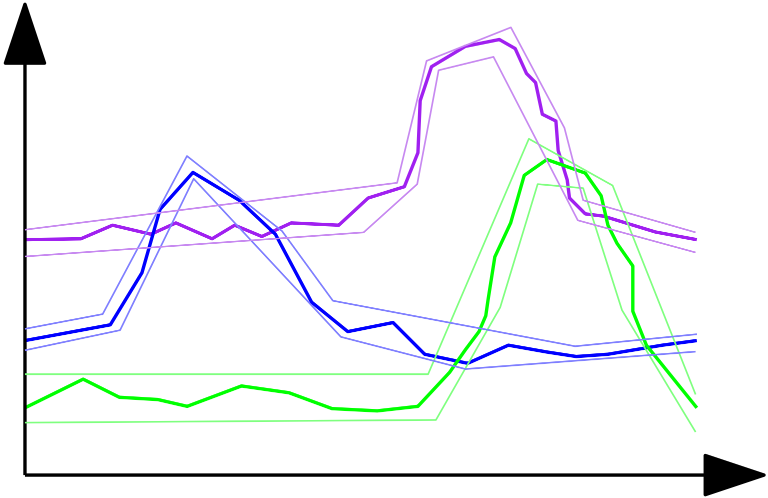
Pruning

Very cheap operations on minimum/maximum of TTFs



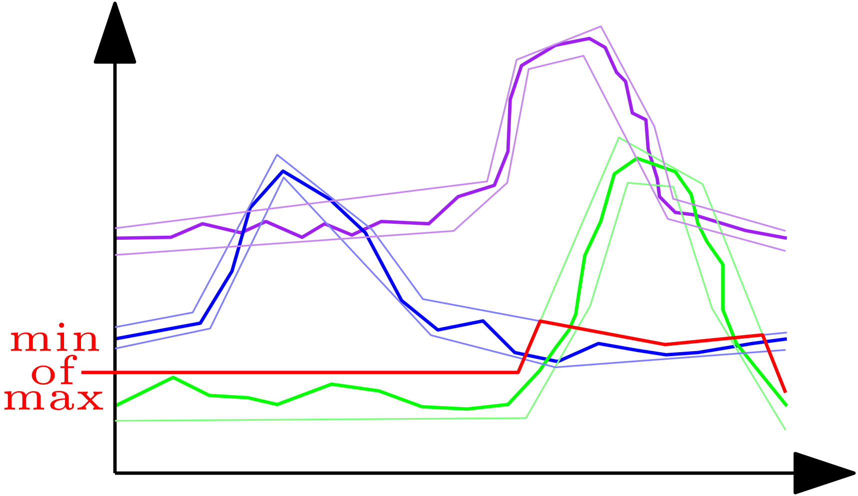
Pruning

Cheap operations on lower/upper bounds of TTFs



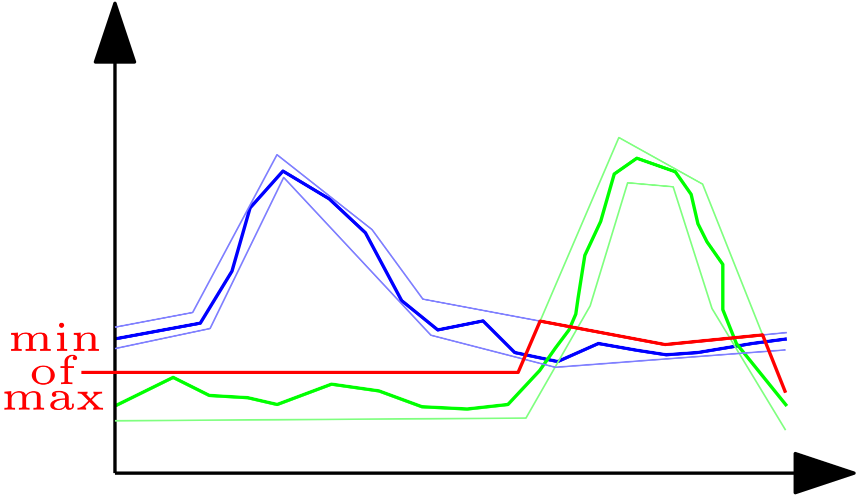
Pruning

Cheap operations on lower/upper bounds of TTFs



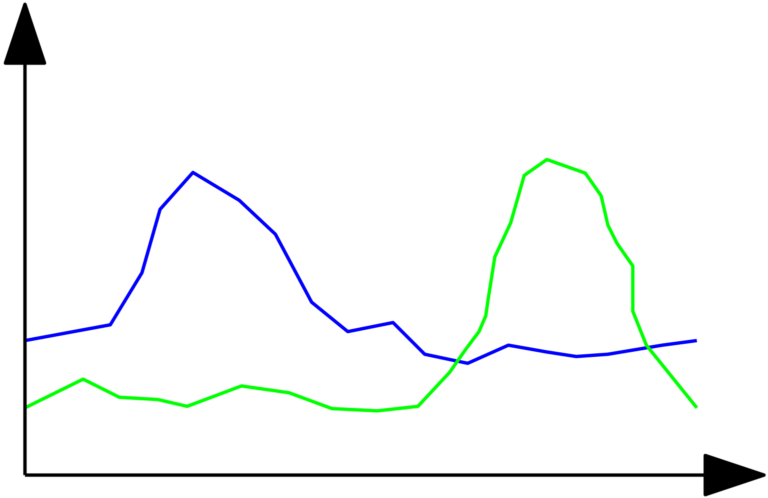
Pruning

Cheap operations on lower/upper bounds of TTFs



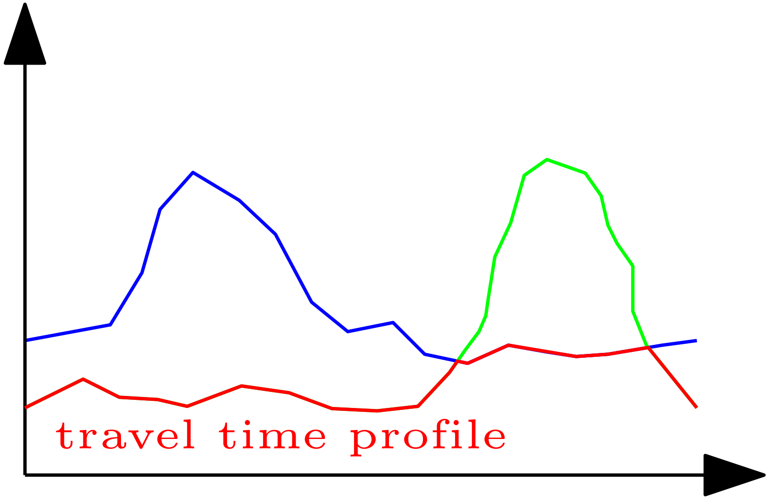
Pruning

Expensive operations on exact TTFs



Pruning

Expensive operations on exact TTFs



5 Algorithms

Enhancing Intersect

- INTERSECT
- MINCANDIDATE computes upper bound using $c_{\min}(s,t) := \operatorname{argmin}_u \left\{ \min g_u + \min f_u \mid (u, f_u) \in F_s, (u, g_u) \in B_t \right\}$
  $\Rightarrow$ especially good for pruning a time query
- RELEVANTCANDIDATE computes the set of candidates where the exact TTF operations are executed.
  $\Rightarrow$ Precomputation only works on lower/upper bounds of TTFs.
  $\Rightarrow$ Drop search space entry of a candidate that is in no set.
- OPTCANDIDATE computes for each departure time an optimal candidate.
  $\Rightarrow$ Precomputation works on exact TTFs but does not store them.
- TABLE uses INTERSECT to computes and stores a whole table.

Error-inducing Approximation

Benefits of approximations:

- require less memory (fewer points)
- are faster to process

We can approximate the TTFs

- on the edges of the TCH
- stored in the forward/backward search
- stored in the table

Definition

An ε-*approximation* is a TTF $f^{\updownarrow}$ with $(1-\varepsilon)f \leq f^{\updownarrow} \leq (1+\varepsilon)f$.

Approximation Guarantees

Problem:

- How to compute an approximation guarantee for a chain of ε-approximations?

Errors stack:

Let $f^{\updownarrow}$ be an ε_f-approximation and $g^{\updownarrow}$ be an ε_g-approximation:

$$f^{\updownarrow} * g^{\updownarrow} : \tau \mapsto f^{\updownarrow}(g^{\updownarrow}(\tau) + \tau) + g^{\updownarrow}(\tau)$$

$\Rightarrow$ Error at evaluation of $g^{\updownarrow}$ is in input of $f^{\updownarrow}$.

Solution: Max. slope α of f: $\forall \tau' > \tau : f(\tau') - f(\tau) \leq \alpha(\tau' - \tau)$.
$\Rightarrow$ $f^{\updownarrow} * g^{\updownarrow}$ is a max $\left\{\varepsilon_f, \varepsilon_g\left(1 + (1 + \varepsilon_f)\alpha\right)\right\}$-approximation of $f * g$.

Experiments

Graph:

- Real-world time-dependent road network of Germany
- 4.7 million nodes, 10.8 million edges
- Midweek (Tue-Thu) traffic with 8% time dependent edges

ε_e [%]	-	0.1	1	10
TCH [MiB]	4 497	1 324	1 002	551

Hardware/Software:

- 2$\times$ Intel Xeon X5550 processors (Quad-Core) @ 2.67 GHz
- 48 GiB of RAM
- GCC 4.3.2

Intersect

size	ε_e [%]	ε_s [%]	ε_p [%]	preprocessing search [s]	preprocessing RAM [GiB]	search space [MiB]	query time [μs]	query profile [μs]
100	-	-	0.1	7.5	6.5	1 639	5.17	1 329
500	-	-	0.1	33.8	13.1	8 228	7.43	1 494
1 000	-	-	0.1	68.0	21.4	16 454	7.97	1 412
1 000	-	-	-	53.1	20.8	15 897	7.99	7 633
1 000	1	-	-	1.5	1.6	349	6.13	108.2
1 000	-	1	-	64.9	5.3	72	6.46	18.4
1 000	0.1	0.1	-	4.7	1.7	189	6.29	52.8
1 000	1	1	-	1.8	1.3	65	5.48	15.1
1 000	10	10	-	0.7	0.9	47	6.34	22.0
10 000	1	1	-	18.2	2.0	650	6.80	16.3

TCH 720 μs time query
32.75 ms exact profile query
2.94 ms approximate profile query ($\varepsilon_e = 1\%$)

MinCandidate

size	ε_e ε_s [%]	preprocessing srch [s]	link [s]	RAM [GiB]	$c_{\min}$ [MiB]	search space [MiB]	query time [μs]	profile [μs]
100	-	6.0	0.0	6.5	1	1 583	3.11	6 941
1 000	-	53.1	0.4	20.8	7	15 897	4.97	7 087
1 000	1	1.8	0.4	1.3	7	65	4.09	13.8
10 000	1	18.2	49.0	2.8	649	650	4.94	14.4

RelevantCandidate

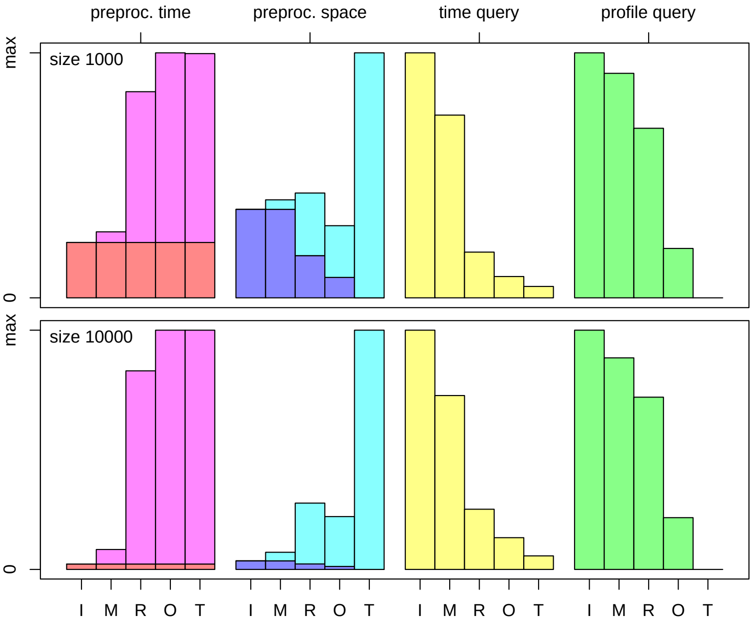
size	ε_e ε_s [%]	ε_p [%]	preprocessing srch [s]	link [s]	RAM [GiB]	c_{rel} [MiB]	search space [MiB]	query time [μs]	profile [μs]
100	-	0.1	7.5	0.3	6.5	1	246	0.72	1 202
1 000	-	0.1	68.0	59.7	35.6	32	3 565	1.29	1 412
1 000	-	1	67.4	14.2	23.9	34	4 195	1.36	2 032
1 000	-	-	53.1	6.1	20.9	49	9 343	2.00	7 651
1 000	1	-	1.8	5.0	1.4	46	31	1.02	10.5
10 000	1	-	18.2	651.3	9.3	4 605	415	1.71	11.7

OptCandidate

size	ε_e ε_s [%]	ε_p [%]	preprocessing srch [s]	link [s]	RAM [GiB]	c_{opt} [MiB]	search space [MiB]	query time [μs]	profile [μs]
100	-	0.1	7.5	2.1	6.5	1	241	0.49	1 168
500	-	0.1	33.8	53.7	13.1	10	1 608	0.73	1 391
1 000	-	0.1	68.0	213.8	21.4	39	3 489	0.81	1 332
1 000	0.1	-	4.7	11.5	1.8	39	42	0.54	9.8
1 000	1	-	1.8	6.3	1.4	38	15	0.48	3.0
1 000	10	-	0.7	7.6	1.0	62	19	0.49	5.7
10 000	1	-	18.2	788.3	7.7	3 775	226	0.90	3.5

Table

size	ε_e [%]	ε_s [%]	ε_p [%]	ε_t [%]	preprocessing srch [s]	link [s]	RAM [GiB]	table [MiB]	query time [μs]
100	-	-	0.1	-	7.5	1.9	7.6	1 086	0.25
500	-	-	0.1	-	33.8	58.5	45.7	27 697	0.42
500	-	-	-	-	26.6	266.7	45.5	27 697	0.42
500	1	-	-	-	0.8	4.8	1.9	427	0.26
1 000	1	-	-	-	1.5	19.0	3.6	1 689	0.32
1 000	1	1	-	-	1.8	6.3	1.6	180	0.25
1 000	-	-	0.1	1	68.0	298.2	21.5	110	0.25
1 000	0.1	0.1	-	0.1	4.7	12.3	2.1	270	0.26
1 000	1	1	-	1	1.8	6.7	1.5	94	0.23
1 000	10	10	-	10	0.7	7.1	1	76	0.22
10 000	1	1	-	1	18.2	815.2	17.8	9 342	0.38



$\varepsilon_e = 1\%$
$\varepsilon_s = 1\%$

Errors

ε_e [%]	1	-	0.1	1	10	-	0.1	1	10
ε_s [%]	-	1	0.1	1	10	-	0.1	1	10
ε_t [%]	-	-	-	-	-	1	0.1	1	10
avg. [%]	0.08	0.12	0.01	0.18	2.1	0.17	0.02	0.30	3.1
max. [%]	0.89	0.98	0.17	1.75	16.9	1.00	0.27	2.66	24.9
theo. [%]	2.07	1.44	0.35	3.55	41.0	1.00	0.45	4.58	55.1

Conclusion

- Computation of search spaces gives linear algorithm INTERSECT.
- Computation of data additional decreases query times and can also decrease space.
- Approximate TTFs significantly reduce time and space.
- Guaranteed error bounds based on max. slope.

Thanks for your attention.

Any question?